uc3m | Universidad Carlos III de Madrid

Grado en Ingeniería Electrónica Industrial y Automática
2016/2017

Trabajo Fin de Grado

REAL-TIME 3D MAP RECONSTUCTION USING MONOCULAR AERIAL IMAGES

Sergio Campos Novoa

Tutor/es
Abdulla Al-Kaff
2017-10-02

Agradecimientos

Me gustaría agradecer a todo Laboratorio de Sistemas Inteligentes de la Universidad Carlos 3 de Madrid y más concretamente a mi tutor Abdulla Al-Kaff por la acogida y la ayuda otorgada.

También me gustaría agradecer a mi familia, que ha trabajado mucho para que yo pueda estudiar en la universidad. A mis compañeros y amigos Armando, Carlos y Roberto porque estos cuatro años se han hecho muy divertidos y porque no estaría aquí en este momento sin su ayuda y apoyo. Y en especial, a mi pareja Iris, porque a su lado he aprendido quien soy y quien quiero ser.

Resumen

Los drones tienen una gran importancia en el desarrollo tecnológico de nuestra vida actual, su gran movilidad les permite trabajar en entornos complicados o aportar mejores puntos de vista. Además, la capacidad de portar una pequeña cámara de gran calidad presenta una gran sinergia con el campo de la visión por computador.

En esta memoria se propone un análisis del estado del arte actual sobre la recreación de entornos en tres dimensiones y se explicarán los conceptos teóricos básicos. Se seleccionará y se explicará el funcionamiento en profundidad de uno de los proyectos pertenecientes a dicho estado del arte que servirá como punto de partida. Se propondrán las siguientes mejoras: una guía práctica que facilite el uso para conjuntos propios de imágenes, un algoritmo que actúe sobre cada punto de forma individual y obtenga su color original de una de sus proyecciones para generar una nube de puntos a color y se estudiará la distancia relativa obtenida por el algoritmo en busca de una relación para transformar esa distancia en verdadera magnitud.

Índice general

Índice de figuras

Capítulo 1

Introducción

La generación de entornos tridimensionales proviene históricamente de la esteoroscopía, que recrea la ilusión de la profundidad a partir de dos imágenes bidimensionales. Esta técnica ha interesado a la especie humana desde la antigüedad, pero el primer inventor de una pantalla estéreo o pantalla 3D fue Sir Charles Wheatstone en 1838 para demostrar la importancia de la percepción de la profundidad debida a nuestra visión binocular [4]. Desde entonces, los sistemas de visión estereoscópica han avanzado centrándose mayormente en el entretenimiento audiovisual, pero actualmente no sólo tenemos la capacidad de crear una ilusión de profundidad, sino que podemos calcular dicha profundidad y expresarla mediante un entorno tridimensional. Muchos campos distintos como la medicina, la construcción o la industria audiovisual están interesados en esta tecnología, pero por encima destaca el de la Robótica.

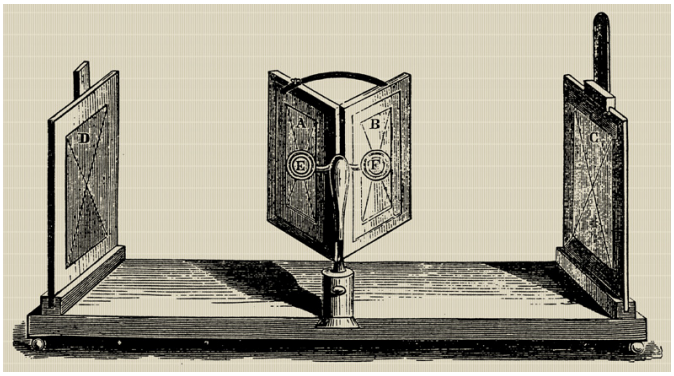

Figura 1.1: Espejo estereoscópico creado por Charles Wheatstone.

El visual SLAM (del inglés, "Simoultaneous Localization And Mapping") es una técnica usada con robots y vehículos autónomos que reconstruye entornos desconocidos en tres dimensiones mientras calcula y rastrea su propia posición [5]. Para reconstruir el entorno tridimensional necesitamos conocer la profundidad de los puntos de la imagen utilizando la disparidad y la geometría estéreo. Sin embargo, nuestro sistema consta únicamente de una cámara monocular, por lo que simularemos un sistema estéreo con dos imágenes de la misma cámara en distintos momentos temporales. Para realizar dicha simulación necesitaremos conocer la diferencia de posición entre esas dos imágenes y, por lo tanto, estaremos rastreando

continuamente la posición de la cámara.

Por otra parte, los Vehículos Aéreos No Tripulados (VANT), o simplemente drones, comienzan su desarrollo durante la Primera Guerra Mundial y la Segunda Guerra Mundial utilizados como blancos móviles para entrenamiento. No es hasta finales del siglo XX cuando realmente se aprovechan sus principales cualidades: Capacidad de uso en áreas de alto riesgo o difícil acceso y no requiere la presencia del piloto.

Hoy en día, los drones poseen otras cualidades nuevas como la gran movilidad (sobretodo en el eje vertical) y una amplia variedad de configuraciones que los convierten en un sistema idóneo para las aplicaciones de la visión por computador, como el visual SLAM.

1.1. Objetivos

Este proyecto pretende reutilizar un algoritmo de reconstrucción de entornos ya existente, adaptarlo a nuestro sistema formado por un dron y una cámara monocular, (un sistema más sencillo y fácil de replicar) y mejorarlo, con un coeficiente para medir la distancia recorrida por el dron y la asignación de los puntos del mapa a su color original.

Se obtendrá un factor de corrección para obtener la distancia real a partir de la posición adimensional proporcionada por el algoritmo y se analizará la información de cada punto perteneciente a la nube de forma individual para encontrar el color que le pertenece en la imagen original.

1.2. Estructura del trabajo

Capítulo 2. Estado del Arte:

Se comentarán los últimos de SLAM visual con cámara monocular y se explicarán de forma concisa sus diferentes puntos de vista y sus aproximaciones.

También aparecerán reflejados los objetivos principales de cada uno de ellos, tanto los que tienen en común, como los que no.

Capítulo 3. Matemática del SLAM:

Se explicará la geometría básica detrás de la generación de entornos dimensionales con un sistema estéreo. Se explicará la adaptación de dicho sistema estéreo a la cámara monocular propuesta por nuestro sistema.

Se analizará y se explicará en profundidad el algoritmo DSO ("Direct Spare Odometry") a partir del cual comenzaremos a trabajar. Se explicarán las aportaciones al modelo.

Capítulo 4. Resultados:

Se enumerará y explicará la plataforma con la que se ha realizado el proyecto y se comentarán todos los experimentos realizados en la implementación del

algoritmo, la obtención de la distancia real y la asignación del color original de la nube de puntos.

Capítulo 5. Conclusiones:

Se comentarán los resultados obtenidos en los experimentos y cómo afectan a nuestro proyecto.

Capítulo 6. Trabajos futuros:

A partir de las conclusiones, se propondrán trabajos a realizar a partir de este o que se puedan beneficiar del mismo.

Capítulo 2

Estado del Arte

2.1. Scale Drift-Aware Large Scale Monocular SLAM

El proyecto se presenta como la primera variante monocular a su estado del arte, en el que se habían desarrollado sistemas de SLAM visual capaces de trabajar de forma precisa, a gran escala y en tiempo real que requerían de visión estéreo. La principal motivación es que las cámaras monoculares son más baratas, compactas y fáciles de calibrar [6].

Una única cámara moviéndose por un escenario estático puede proveer de la geometría estéreo utilizando dos imágenes de la misma escena en distintos momentos temporales y desde disntintos ángulos. Debido a que no se conoce la distancia exacta entre ambos frames (al contrario que la visión estéreo), sólo se puede conocer la distancia en el mapa de forma relativa.

En este proyecto se describe un SLAM visual nuevo en relación a su estado del arte que casi funciona en tiempo real en grandes entornos de la vida real con una cámara monocular. Se le da una gran importancia a optimización del algoritmo de estimación de la posición de la cámara y el del cierre del recorrido.

2.2. LSD-SLAM

El LSD SLAM es un algoritmo creado por Jakob Engel y Daniel Cremers de la Universidad Técnica de Zurich con un nuevo enfoque en la reconstrucción de entornos en tres dimensiones. En comparación al estado del arte de los SLAM visuales con métodos directos de su momento, LSD-SLAM no utiliza las características de las vecindades de los puntos clave (keypoints) para emparejar los puntos de dos imágenes distintas. También es capaz de generar mapas tridimensionales semidensos, consistentes y de gran escala e en tiempo real [7].

Como novedad, incluye un algoritmmo probabilístico para incorporar el ruido de la profudidad relativa estimada en la detección de la poscición de nuevas imágenes clave o de referencia (keyframes).

2.3. ORB-SLAM

Este proyecto se presenta como una opción más robusta en comparación con su estado del arte. Es un algoritmo de SLAM visual y monocular que en comparación al proyecto anterior, vuelve a utilizar un método indirecto bastado en el algoritmo ORB para la detección de puntos clave y su correspondencia a lo largo de varios frames [8].

ORB es un descriptor presentado por Ethan Rublee, Vincent Rabaud, Kurt Konolige and Gary R. Bradski en 2011 como alternativa a otros descriptores como SHIFT o SURF en coste computacional. Se basa en una mezcla del algoritmo FAST para encontrar keypoint y el descriptor BRIEF que tiene un funcionamiento bastante pobre con las rotaciones. ORB se encarga de mejorar dicho funcionamiento discretizando el ángulo en treinta incrementos, y con una tabla de consulta (lookup table) con patrones BRIEF se busca el resultado más consistente de todos los incrementos y se rota la imagen en ese ángulo antes de utilizar el descriptor BRIEF.

El algoritmo ORB SLAM también se centra en los resultados en grandes entornos, en la relocalización en tiempo real para recuperarse de un fallo en el seguimiento de la cámara y en la precisión para cerrar el recorrido comprobándolo en cada frame y luego esparciendo el error por el resto del mapa.

2.4. Direct Sparse Odometry (DSO)

El algoritmo DSO es la segunda iteración de Jakob Engel y Daniel Cremers de la Universidad Técnica de Zurich de recreación de entornos en tres dimensiones con una cámara monocular. Esta vez también realizan una aproximación directa centrándose en el error fotométrico de cada punto en las vecindades de los píxeles [3].

En cuanto a la densidad de la nube de puntos optan por un método disperso, que se caracterizan por centrarse en entornos más pequeños y definidos como esquinas y bordes. Esto permite ejecutarlo en tiempo real debido al menor coste computacional a costa de perder precisión y atractivo.

Debido a que este algoritmo es el más actual y en su memoria presentan buenos resultados ha sido elegido como punto de partida para la realización de este proyecto. EL funcionamento en profundidad de este algoritmo se realizará más adelante.

Capítulo 3

Teoría del SLAM

Las cámaras digitales transforman las coordenadas tridimensionales del mundo en real en coordenadas bidimensionales de la imagen [9].

$$(x, y, z)^T \rightarrow (u, v)^T \tag{3.1}$$

Podemos definir **P** como la matriz de proyección de perspectiva que realiza la transformación de 3D a 2D, pero como no es posible aplicar la transformación por el tamaño de la matriz, recurrimos a añadir un componente extra.

$$M = (x, y, z, 1)^T \rightarrow u = (U, V, S)^T \tag{3.2}$$

Siendo:

$$u = \frac{U}{S},\ v = \frac{V}{S}\ y\ S \neq 0 \tag{3.3}$$

La matriz de proyección de perspectiva **P** puede descomponerse como el producto de dos matrices. Una matriz **A** de parámetros intrínsecos de la cámara y una matriz **G** de parámetros extrínsecos.

Los parámetros intrínsecos hacen referencia a las características de la cámara. Este proyecto cuenta con una cámara monocular basada en el modelo **pinhole**, que reduce la óptica de la imagen a un punto(foco) por el que pasan todos los rayos luminosos antes de reflejarse en el sensor de la cámara.

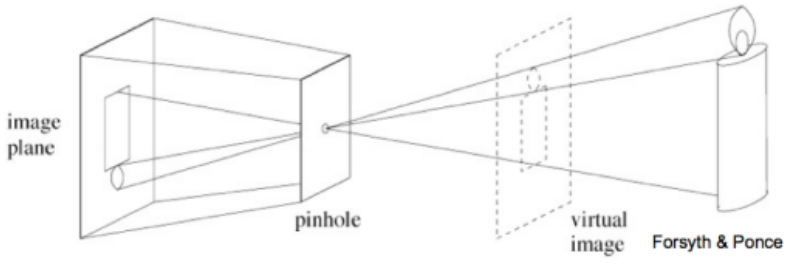

Figura 3.1: Esquema del modelo pinhole.

La distancia entre entre el foco y el plano de proyección de la imagen se denomina distancia focal y es muy útil para obtener medidas en el mundo real a partir de una imagen.

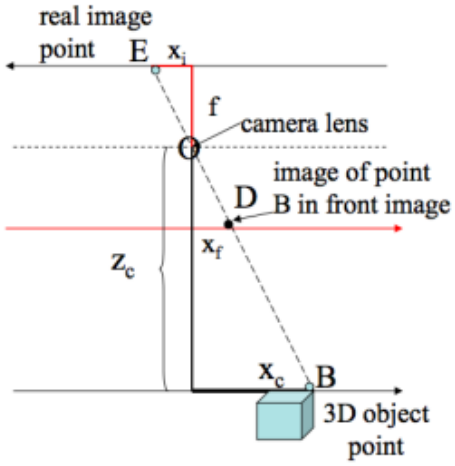

Figura 3.2: Esquema de la relación de magnitudes en el eje X.

Por lo tanto podemos obtener las siguientes relaciones

$$\frac{x_i}{f} = \frac{x_c}{z_c}\ ,\ \frac{y_i}{f} = \frac{y_c}{z_c} \tag{3.4}$$

Utilizando la nomenclatura de la ecuación 3.3:

$$\frac{u}{f} = \frac{x}{z}\ ,\ \frac{v}{f} = \frac{y}{z} \tag{3.5}$$

Se puede expresar de la siguiente manera:

$$\begin{bmatrix} U \\ V \\ S \end{bmatrix} = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{3.6}$$

Donde la matriz central representa los parámetros intrínsecos básicos de la cámara. Tendremos en cuenta si los píxeles de la cámara no son cuadrados con f_u y f_v y también si el plano de la imagen está desplazado con respecto al plano de la cámara con u_o y v_o. Al final tenemos:

$$A = \begin{bmatrix} f_u & 0 & u_o \\ 0 & f_v & v_o \\ 0 & 0 & 1 \end{bmatrix} \tag{3.7}$$

Los parámetros extrínsecos de la cámara representan la posición de los ejes de la cámara en relación con los ejes del mundo. La tansformación se expresa como una sencilla matriz de traslación y rotación:

$$G = \begin{bmatrix} r_{00} & r_{01} & r_{02} & t_x \\ r_{10} & r_{11} & r_{12} & t_y \\ r_{20} & r_{21} & r_{22} & t_z \end{bmatrix} \tag{3.8}$$

Y la transformación final:

$$\begin{bmatrix} U \\ V \\ S \end{bmatrix} = \begin{bmatrix} f_u & 0 & u_o \\ 0 & f_v & v_o \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{00} & r_{01} & r_{02} & t_x \\ r_{10} & r_{11} & r_{12} & t_y \\ r_{20} & r_{21} & r_{22} & t_z \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{3.9}$$

El único elemento que falta es la profundidad, y todos los algoritmos visuales de generación de entornos en tres dimensiones parten de la visión esteoroscópica, una técnica que crea una ilusión de profundidad utilizando, entre otras cosas, dos imágenes, lo que ha derivado en la visión estéreo por ordenador.

3.1. Visión estéreo por ordenador

La visión estéreo por ordenador se basa en la extracción de la información tridimensional de imágenes digitales comparando la misma escena desde dos puntos de vista distintos.

Tradicionalmente es un sistema compuesto por dos cámaras desplazadas horizontalmente de forma similar a la visión binocular de los seres humanos, y la profundidad relativa puede obtenerse de un mapa de disparidad y el modelo se explica matemáticamente utilizando la geometría epipolar.

La geometría epipolar es la rama de las matemáticas que se dedica a estudiar las relaciones geométricas de los puntos tridimensionales del espacio y sus proyecciones en imágenes 2D obtenidas desde distinto punto de vista [10].

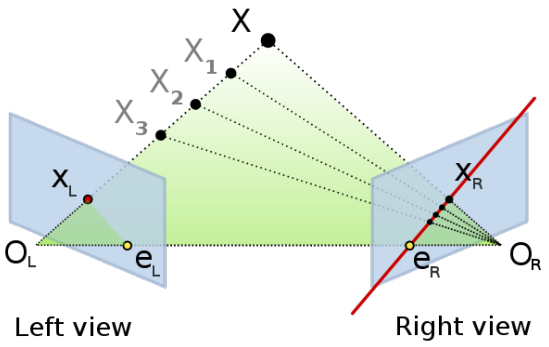

Figura 3.3: Esquema de la geometría epipolar.

Utilizando la nomenclatura de la imagen superior tenemos O_L como foco de la imagen izquierda y O_R como foco de la imagen derecha.

Los puntos e_L y e_R son los epipolos, y se conocen como la proyección del foco de una imagen en la otra.

La proyección X_L del punto tridimensional X corresponde a una dirección en la imagen de la derecha, esta dirección pasa por el foco de la cámara (O_L/e_R) y sirve para encontrar la proyección X_R, se llama línea epipolar, cada proyección genera una en la otra imagen y todas pasan por los epipolos.

Teniendo en cuenta toda esta información, se puede conocer si dos proyecciones pertenecen al mismo punto trazando sus lineas epipolares, utilizando las matrices esencial y fundamental. La matriz esencial presenta la información de la posición

de ambas cámaras en términos de rotación y traslación **(Ecuación 3.8)** , sin embargo, la matriz fundamental además añade información sobre la distancia focal **(Ecuación 3.9)**. La matriz fundamental simplfica este modelo ya que $F^T * X_l$ corresponde a linea epipolar formada por e_R y X_R.

La disparidad binocular hace referencia a diferencia de posición de un objeto visto por el ojo izquierdo y el derecho resultado de su separación horizontal (paralaje). Después el cerebro utiliza esta diferencia de posición para percibir la profundidad de los objetos.

En visión por computador, la disparidad hace referencia a la diferencia de coordenadas de un punto determinado en dos imágenes estéreo de una misma escena. Y también podemos utilizar esta información para calcular una profundidad relativa.

El sistema estéreo ideal presenta dos cámaras iguales, con la misma distancia focal, con los ejes ópticos colineales, es decir los planos de las imágenes de las cámaras corresponden al mismo plano, y distancia entre focos conocida.

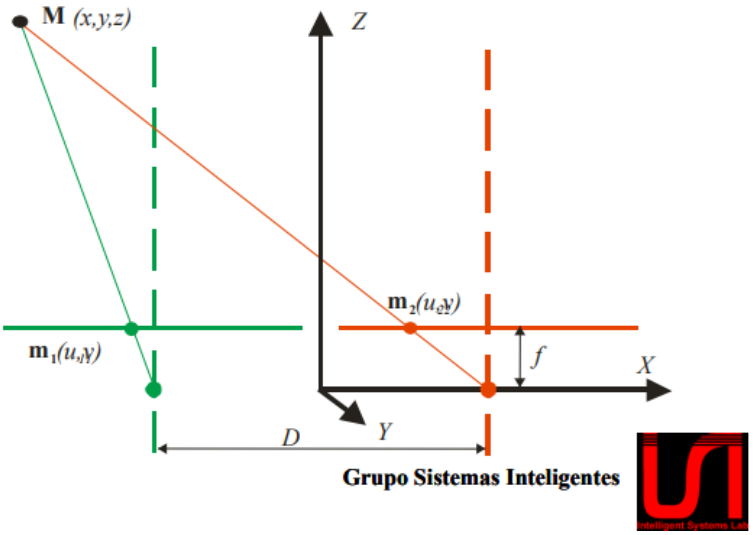

Figura 3.4: Ejemplo de disparidad en eje X de dos cámaras paralelas. Fuente: LSI.

Siendo $M = (x, y, z)$ el punto tridimensional, m_1 y m_2 sus proyección en el sistema estéreo, D la distancia horizontal entre ambas cámaras con el eje de coordenadas situado en el centro y f la distancia focal.

$$\frac{u_1}{f} = \frac{x + \frac{D}{2}}{z} \ , \ \frac{u_2}{f} = \frac{x - \frac{D}{2}}{z} \tag{3.10}$$

$$x = z\frac{u_i}{f} - \frac{D}{2} = z\frac{u_d}{f} + \frac{D}{2} \tag{3.11}$$

$$z = f\frac{D}{u_i + u_d} = f\frac{D}{d} \tag{3.12}$$

Siendo d la disparidad del punto que es inversamente proporcional a la distancia.

Como este sistema está alineado horizontalmente, los epipolos están en el infinito, y todas las lineas epipolares son horizonrales, ergo, todos los puntos tridimensionales tienen su proyección vertical idéntica en ambas imágenes (y_2 e y_1).

La rectificación de las imágenes estéreo se basa en reproyectar los planos de ambas imágenes de forma que pertenezcan al mismo plano paralelo a la línea que une los centros de las cámaras de la siguiente forma:

Figura 3.5: Ejemplo de rectificación de sistema estéreo. Fuente: University of Illinois

De forma que tengan las misma propiedades que las condiciones ideales de la Figura 3.4

3.1.1. Adaptación al sistema monocular

En este sistema vamos a utilizar una cámara monocular, por lo que vamos a tener que simular un sistema estéreo utilizando dos frames de la cámara en distintos momentos temporales que muestren la misma escena desde distintos puntos de vista. Al ser la misma cámara se cumplen dos de las condiciones ideales (cámaras iguales y misma distancia focal), no tendrá los ejes ópticos colineales y se calculará la distancia entre ambos focos mediante un algoritmo de "pose estimation"llamado Homografía.

3.1.2. Calibración

Antes de empezar a utilizar la cámara realizaremos un proceso de calibración muy común en la visión por computador. Ya se ha explicado el modelo pinhole **(Figura 3.1)** anteriormente, pero hay que añadir que es uno de los más famosos debido a su coste, pero por contrapartida nos genera una considerable distorsión radial y tangencial [11].

Para corregir esta distorsión se utiliza un patrón claramente definido tanto en el mundo real como en la imagen, el más común es el tablero de ajedrez debido a que tiene dos colores fácilmente identificables en la imagen con altos gradientes en los bordes y sabemos que son cuadrados perfectos e idénticos.

El paquete de ROS "*camera_calibration*" nos provee de una herramienta donde introduciremos el tamaño de nuestro tablero en esquinas interiores y en centímetros cuadrados $[cm^2]$ y el topic donde se está publicando la imagen.

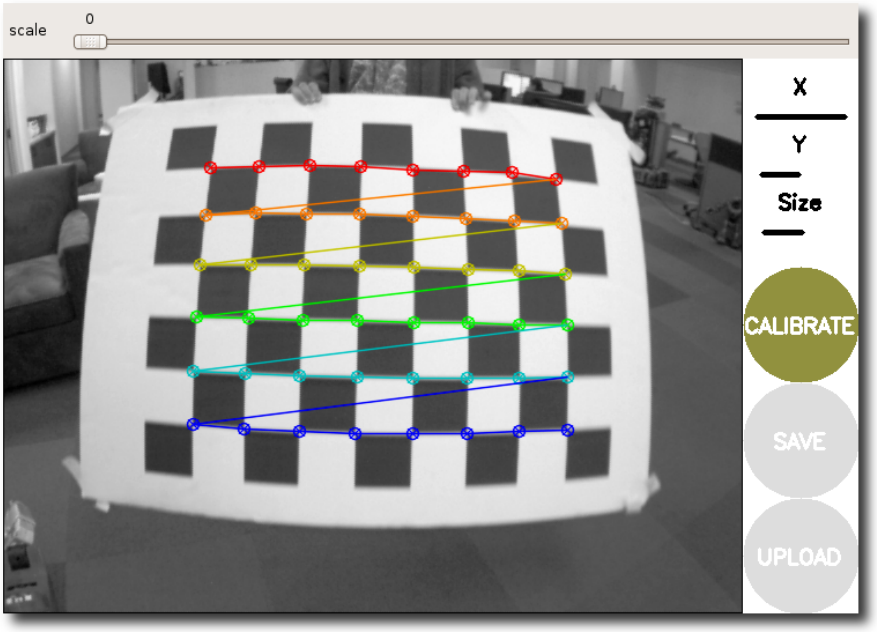

Figura 3.6: Paquete de calibración de cámaras en ROS. (Fuente: [1])

Una vez empieze el programa deberemos rotar y desplazar el tablero en todas las direcciones y por todo el plano de la imagen para que se recojan todos los datos necesarios para realizar la calibración correctamente. Al terminar, obtendremos la matriz esencial de la cámara y una serie de coeficientes para corregir la distorsión, en nuestro caso:

$$A = \begin{bmatrix} 376{,}4 & 0 & 308{,}63 \\ 0 & 376{,}43 & 238{,}27 \\ 0 & 0 & 1 \end{bmatrix} \; y \; D = \begin{bmatrix} -0{,}318 & 0{,}10674 & 0{,}0001 & -0{,}00033 & 0 \end{bmatrix} \tag{3.13}$$

3.1.3. Homografía

La homografía se define en visión por computador como la relación en términos de rotación y tralación de dos imágenes que representan la misma escena desde dos puntos de vista. Este algoritmo necesita un grupo de puntos que aparezcan en ambas imágenes y que estén en el mismo plano. Caculamos la homografía de la siguiente manera [12]:

$$ {}^{a}p_i = \frac{{}^{b}z_i}{{}^{a}z_i} \cdot H_{ba} \cdot K_b^{-1} \cdot {}^{b}p_i \tag{3.14} $$

Donde ${}^{a}p_i$ y ${}^{b}p_i$ son las proyecciones del punto **P** en ambas imágenes **(A y B)**, ${}^{a}z_i$ y ${}^{b}z_i$ son las distancias de los focos de cada imagen al plano que contiene los puntos Y K_a Y K_b son las matrices de parámetros intrínsecos de cada cámara, en nuestro caso es la misma.

3.1.4. Puntos Clave

Para aplicar la geometría epipolar y para encontrar la posición de la cámara en distintos momentos temporales necesitamos conocer una serie de puntos que aparezcan en las dos imágenes. El primer paso es buscar características en la imagen, como detectores de bordes y esquinas, y después clasificarlos con un descriptor.

Encontrar características en las imágenes ("features", en inglés) es una cualidad inherente al ser humano, desde pequeños jugamos con figuras geométricas y somos capaces de encontrar patrones para encajarlas. Pero es más complicado para un ordenador, intentaré explicarlo utilizando la siguiente imagen:

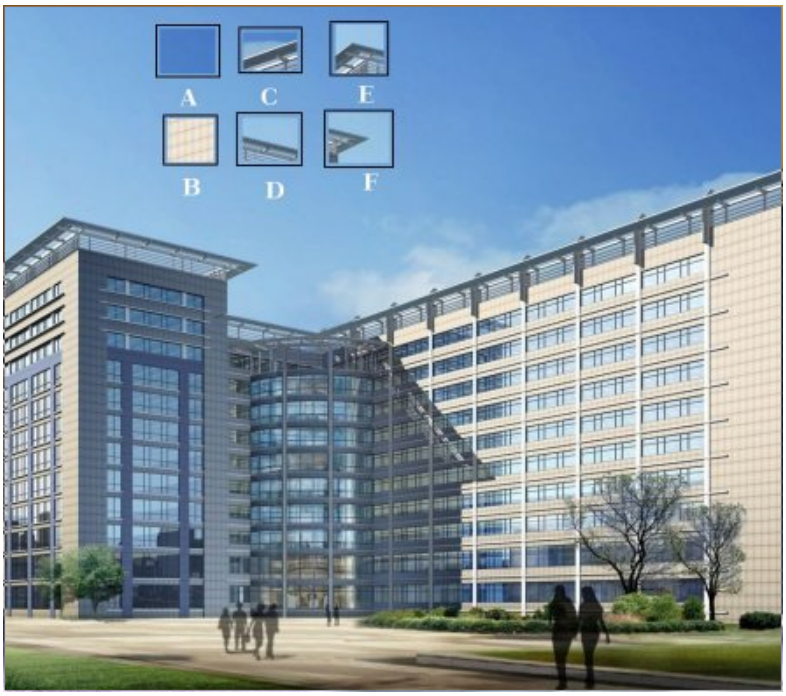

Figura 3.7: Caracteristicas de las imágenes.

Encontrar las partes **A** y **B** en la imagen original es muy complicado debido a que son patrones repetitivos que aparecen por amplias superficies en la imagen. Las partes **C** y **D** son más sencillas porque pertenecen al **borde** de un edificio y las más sencillas serían **E** y **F** porque corresponden a las **esquinas**. Los bordes y las esquinas son **características** fáciles de encontrar en una imagen porque hay grandes cambios en el valor de la intensidad de pixel entre las zonas separadas por el borde y se crea un gradiente remarcable (incluso en escala de grises).

Una vez se han localizado las zonas con mayor potencial para albergar keypoints, se utilizan algoritmos para filtrar los de menor importancia utilizando umbrales:

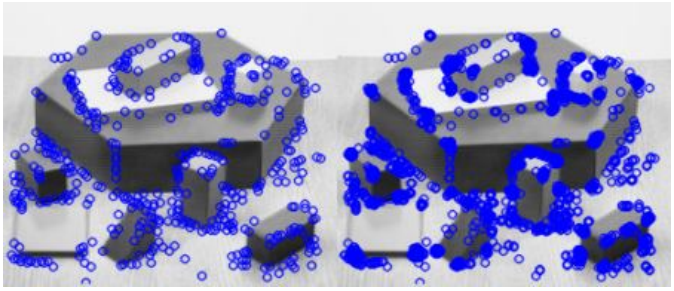

Figura 3.8: Keypoints detectados en bordes y esquinas por FAST.

Después de detectar los puntos, se utilizan descriptores para identificarlos lo mejor posible y detectarlos a lo largo de varias imágenes. Estos descriptores suelen utilizar valores obtenidos de las vecindades de los keypoints como gradientes, intensidades o histogramas.

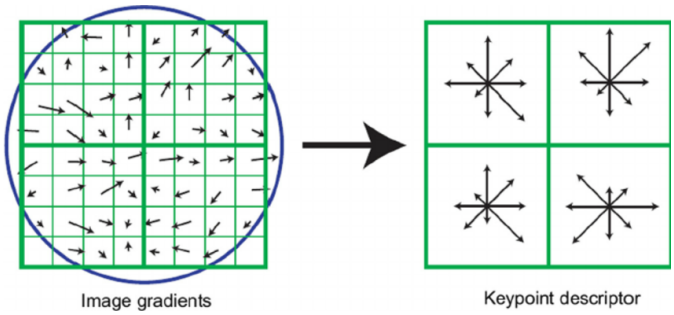

Figura 3.9: Análisis del gradiente en las vecindades de un keypoint.

La detección de puntos en dos imágenes (matching, en inglés) compara los descriptores de los puntos entre sí y empareja los más similares, una vez ya hayamos realizado la homografía, podríamos utilizar la geometría epipolar para reducir el rango de comparación de puntos.

Una vez tengamos relacionados un grupo de puntos en dos imágenes, podremos hallar la homografía (Ecuación 3.14) y la matriz de parámetros extrínsecos (Ecuación 3.8) para hallar la profundidad relativa.

3.2. DSO: Direct Sparse Odometry

El algoritmo DSO se proclama como una nueva formulación directa y dispersa de la odometría visual. Combina el modelo probabilístico buscando reducir el error fotométrico con una optimización de todos los parametros del modelo, incluida la profundidad y el movimiento de la cámara. Este método no depende de algoritmos de búsqueda de keypoints ni de sus descriptores. Se ejecuta en tiempo real omitiendo la suavidad de acabado de otros modelos directos. También incluye un sistema propio de calibración fotométrica y supera a su estado del arte (directos e indirectos) utilizando un dataset bastante variado.

3.2.1. Modelo directo y disperso

Los métodos indirectos de odometría visual, que se define como el proceso de determinar la posición y orientación analizando imágenes de una cámara en movimiento, se diferencian de los modelos directos en que éstos buscan determinadas características de la imagen para obntener sus keypoints y los métodos directos utilizan directamente el valor de intensidad de los sensores en un modelo pobabilístico para buscar keypoints.

Por otra parte, los modelos dispersos se centran en reconstruir solamente grupos independientes de puntos; sin embargo, los modelos directos buscan reconstruir toda la imagen bidimensional en tres dimensiones. Existe una aproximación intermedia ("semidensos") que reconstruye grupos independientes de puntos, pero también busca conectar estos grupos en estructuras mayores. La principal razón de uso de estos modelos a pesar de su peor acabado es la posibilidad de ejecutarlos en tiempo real.

3.2.2. Calibración

La calibración fotométrica es muy importante para este algoritmo debido a que trabaja con el valor de la intensidad de los pixeles en contraste con los detectores de keypoints que son bastante robustos contra la variaciones fotométricas [2].

El formato de calibración utilizado es el mismo que tienen los datasets de imágenes "The TUM monoVO datasetçreado por los mismos autores.

Utiliza una calibración geométrica genérica de una cámara con modelo pinhole para obtener la distancia focal, y los coeficientes de distorsión (Ecuación 3.13).

Incluye una calibración fotométrica basada en el tiempo de exposición en milisegundos y la irradiancia de los sensores. Utiliza alrededor de mil imágenes con diferentes tiempos de exposición (desde 0.05 ms a 20 ms) que cubre una gran parte de los niveles de gris y como resultado ganamos información para los niveles bajos y altos y reducimos el efecto de las alteraciones de iluminación.

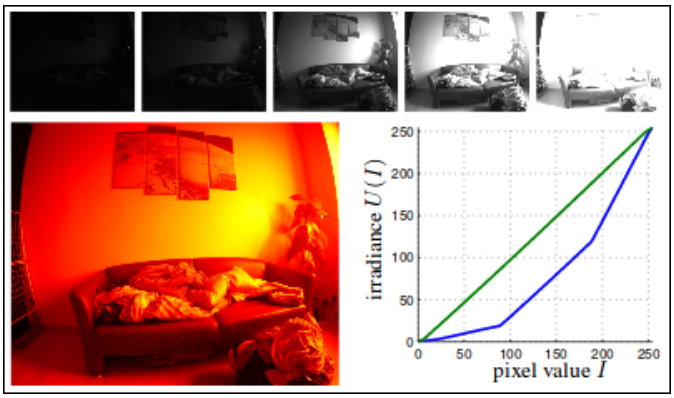

Figura 3.10: Ejemplo de calibración fotométrica utizando la irradiancia. (Fuente: Monocular Visual Odometry Dataset [2])

También utiliza una calibración de viñeteado (en inglés, vignetting) para centrarse en la imagen y reducir el número de puntos generados en la zona de la imagen donde puede aparecer más distorsión generada por el modelo pinhole. Utiliza un patron reconocible sobre un fondo uniforme (preferiblemente blanco) y lo observa desde todos los ángulos posibles y obtiene una imagen con zonas más oscuras que indican un factor de atenuación para la imagen original:

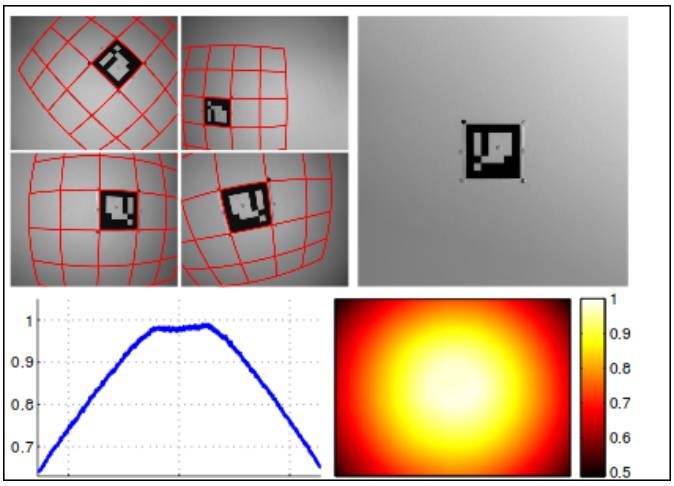

Figura 3.11: Ejemplo de calibración de viñeteado. (Fuente: Monocular Visual Odometry Dataset [2])

En la parte superior derecha, se observa el patrón sobre el fondo blanco, a su izquierda cuatro ejemplos de las distintas posiciones necesarias para la calibración y en la parte inferior aparece el factor de atenuación de la irradiancia que se va a aplicar más tarde en la imagen original.

3.2.3. Administración de imágenes

Este método utiliza un número determinado de keyframes activos (7 en el ejemplo por defecto). Cada frame inicialmente se compara con los keyframes activos (paso 1), después se descarta o se usa como un nuevo keyframes (paso 2) y uno antiguo es marginalizado (paso 3).

Paso 1: Cuando el sistema crea un keyframe nuevo, todos sus puntos activos se proyectan en el mapa ligeramente dilatados para generar un mapa semidenso. Cada nuevo frame se compara únicamente con este último keyframe utilizando un algoritmo convencional de alineamiento de imágenes y un modelo de movimiento.

Paso 2: De forma similar a la utilizada en el algoritmo ORB-SLAM mencionado en el estado del arte, la estrategia es coger tantos keyframes como sea posible evitando los redundantes y repartirlos adecuadamente por el espacio. Para determinar un nuevo keyframe utilizamos tres criterios:

-Los nuevos keyframes son necesarios cuando aparecen cambios en el campo de visión. Se utiliza la media cuadrática del flujo óptico del último keyframe al último frame.

-La traslación de la cámara crea oclusiones que se arreglan con más keyframes.

-Siempre que el tiempo de exposición cambia considerablemente se toman nuevos frames.

Estos tres criterios pueden cuantificarse y compararse asignándoles un peso relativo.

Paso 3: La estrategia de marginalización de keyframes tiene tres condiciones:

-Siempre nos quedamos con los dos últimos.

-Cuando un frame tiene menos de un 5 % de sus keypoints visibles en el último frames se marginaliza.

-Cuando hay más de siete keyframes que cumplen estas dos condiciones se utiliza una función diseñada para mantener los keyframes bien distribuidos en el espacio pero con más frecuencia cuanto más se acercan al último keyframe.

Cada vez que se marginaliza un keyframe todos los puntos que pertenezcan a dicho keyframe desparecen del sistema. Pero siguen pudiendo observarse en el algoritmo.

3.2.4. Administración de puntos

El algoritmo DSO utiliza un método directo de detección de los puntos relevantes de la imagen o keyframes. La mayoría de algoritmos directos se centran en utilizar tanta información de la imagen como sea posible, lo que dificulta su funcionamiento en tiempo real, por lo que realizan estimaciones subóptimas antes de tiempo para compensar ignorando las relaciones entre distintos parámetros.

Los experimentos realizados para el dso muestran que los datos de una imagen son altamente redundantes y el beneficio de usar más puntos se estanca rápidamente. Utilizar métodos directos aporta más información sobre texturas suaves y sobre regiones o bordes repetitivas.

El algoritmo intenta trabajar con 2000 puntos activos en el espacio tridimensional esparcidos equitativamente a través del espacio y los frames. En un primer momento, se buscan 2000 puntos candidatos en cada ketframe (paso 1). Los puntos candidatos no se añaden inmediatamente a la optimización, en su lugar se buscan individualmente en el resto de frames y se genera un valor inicial de profundidad que sirve para la inicialización (paso 2). Cuando hace falta introducir nuevos puntos a la optimización, se selecciona un número de puntos a través de todos los keyframes activos para ser introducidos en dicha optimización (paso3).

Paso 1: la estrategia de generación de keypoints del algoritmo busca que estén bien repartidos por la imagen y que tengan el gradiente lo suficientemente alto en su vecindad. El umbral del gradiente se calcula de forma regional en bloques de 32x32 y es directamente proporcional a la media del gradiente en dicho bloque. Después se realiza una división de la imagen en bloques de DxD y se toma el pixel que mayor gradiente tenga en caso de situarse or encima del umbral. En caso de que no haya ninguno, no se toma ningún píxel de ese bloque. Para mejorar el resultado se pueden coger puntos con menor gradiente reduciendo el umbral en zonas en las que no se ha obtenido ningún punto, y obtendríamos información sobre texturas suaves de las superficies o ligeros cambios de iluminación, se puede repetir este proceso reduciendo más el gradiente y aumentando el tamaño del bloque.

Paso 2: los puntos candidatos se localizan en distintos frames con búsqueda discreta recorriendo la linea epipolar, minimizando el error fotométrico. De la mejor coincidencia se calcula una profundidad asociada y su varianza que se utilizará para futuras iteraciones en frames consecuentes.

Paso 3: cuando un grupo de puntos se marginalizan con su frame, un grupo nuevo de puntos candidatos se activan para reemplazarlos. Para mantener una distribución espacial repartida por la imagen, proyectamos todos los puntos activos posibles al keyframe más frecuente y activamos los puntos candidatos que maximizan la distancia a cualquier otro punto existente.

Outlier y detector de oclusiones: Se intentan identificar los outliers tan pronto como sea posible. Se descartan los puntos que no son lo suficientemente distintos de otros utilizando la linea epipolar y se utiliza un umbral de error para eliminar directamente los puntos que queden por encima. Dicho umbral se adapta a la calidad de la imagen de forma directamente proporcional, de forma que se descartan menos observaciones de imágenes con peor calidad.

3.3. Aportaciones al modelo.

3.3.1. Calibración

El algoritmo DSO necesita los siguientes archivos de calibración para funcionar correctamente: “camera.txt”, “pcalib.txt”, “times.txt”, “vignette.png” y “images.zip”. Estos archivos definen el formato de los grupos de imágenes “The TUM monoVO dataset” creado para odometría visual con la intención de convertirlo en un estándar para comparar resultados.

El principal problema que he encontrado en este dataset es la dificultad de adaptación de tus propias imágenes a su formato, así que explicaré su funcionamiento práctico como parte de las mejoras propuestas para facilitar su uso:

- **images.zip:**

  Este fichero comprimido contiene todas y cada una de las imágenes de la cámara que permiten la ejecución del algoritmo. De forma estricta tienen que estar numeradas de forma ascendiente empezando por 0, y todas tienen que tener el mismo número de dígitos.

- **camera.txt:**

```
Pinhole fx fy cx cy 0
in_width in_height
"crop" / "full" / "none" / "fx fy cx cy 0"
out_width out_height
```

Figura 3.12: Estructura del archivo camera.txt de calibración geométrica. (Fuente: DSO [3])

En la primera línea podemos encontrar la calibración geométrica básica de la cámara, f_x, f_y, c_x y c_y corresponderían aproximadamente a los parámetros intrínsecos de nuestra cámara y cambiaríamos el 0 por 1. La única diferencia con los parámetros intrinsecos son las siguientes correcciones:

$$f_x = \frac{f'_u}{in\_width} \ , \ f_y = \frac{f'_v}{in\_height} \tag{3.15}$$

$$c_x = \frac{u'_o + 0{,}5}{in\_width} \ , \ c_y = \frac{v'_o + 0{,}5}{in\_height} \tag{3.16}$$

Siendo *in_width* e *in_height* el ancho y el alto respectivamente de la resolución de la cámara utilizada. Con los valores calculados en apartados anteriores para nuestra cámara (Ecuación 3.13) obtenemos los siguientes resultados:

$$f_x = 0{,}58812 \ , \ f_y = 0{,}7842 \ , \ c_x = 0{,}4822 \ y \ c_y = 0{,}4964 \tag{3.17}$$

En la tercera línea añade un modo de rectificación que genera un modelo pinhole introducido de forma manual ("fx fy cx cy 0") o una opción que automáticamente ajusta la imagen a un rectángulo ("crop"). La opción "full."está principalmente pensada para depuración, manteniendo el campo de visión original y generando unos bordes negros que pueden corromper la ejecución del programa. En la última linea aparece la resolución de salida del la calibración de la imagen que he utilizado la nativa para evitar errores debidos a reescalados.

- **pcalib.txt:**

Contiene un vector de 256 valores mapeando del cero al 255 a su respectivo valor de irradiancia de la misma forma que una tabla de consulta. Se puede obtener haciendo una gran cantidad de fotos a una misma escena variando el tiempo de exposición como aparece en el apartado anterior, o utilizando el archivo de uno de los datasets de ejemplo.

- **times.txt:**

  En este archivo aparecen 3 columnas, en la primera aparecen los nombres de las imágenes enumerados por orden, en la segunda columna se guarda el tiempo en microsegundos correspondiente a cada frame utilizando el formato de "high_resolution_clock" de c++ y en la tercera aparce el tiempo de exposición de la cámara.

  Lamentablemente por limitaciones del hardware no he podido acceder al valor de eel tiempo de exposición ni he podido ajustarlo, por lo que se perderá parte de la precisión de los datasets originales al usar los propios.

- **vignette.png:**

  Este archivo no se ha tenido en cuenta en ningún experimento debido a la falta del programa de calibración en la implementación del código.

3.3.2. Mejora coloreado original del mapa

Esta mejora del programa se centra en asignarle a los keypoints los colores originales en sus respectivas imágenes.

La visualización de la nube de puntos utiliza un software libre llamado Pangolin, donde podemos editar el color de cada punto de forma individual.

El algoritmo DSO genera los puntos tridimensionales mediante la descomposición del vector de la distancia desde el foco de la cámara utilizando la posición bidimensional del píxel de la imagen al que pertenece dicho punto punto. Por lo tanto, teniendo la imagen original y su posición en dicha imagen podemos obtener el código de color RGB de dicho pixel y aplicarselo al punto tridimensional.

También he añadido un algoritmo para generar un archivo en formato ".txt" con información sobre el frame al que pertenece cada punto, la posición bidimensional de su proyección en dicha imagen y sus coordenadas tridimensionales a partir de la misma imagen.

Y por último un programa por separado que toma todos los ficheros generados y necesarios para el algoritmo DSO y obtiene la posición tridimensional con respecto al eje del mundo y el color en RGB para cada punto de la nube y lo guarda en cualquier formato posible de las librerías PCL compatibles con ROS, también podría ser editado de forma sencilla para adaptarse al formato de otros softwares de generación de nubes de puntos.

3.3.3. Factor de conversión para la distancia

El software DSO genera por defecto un archivo con los resultados donde se indica el tiempo, la posición tridimensional y la posición en cuaternios de todos los keyframes seleccionados. Por esto he intentado hallar un coeficiente para transformar la distancia recorrida en metros.

He realizado numerosos experimentos en los que generaba un mapa tridimensional de una distancia conocida en linea recta y lo comparo con la distancia adimensional obtenida por el algoritmo.

Capítulo 4

Resultados

4.1. Plataforma

4.1.1. Hardware

- **Cámara SJ4000 Wifi**

  La cámara deportiva SJ4000 dispone de un objetivo de 12 Megapíxeles con un sensor CMOS, también tiene una lente de gran angular de 170º.

  Permite grabación de video a 720p a 60 frames por segundo y es sumergible hasta unos 30 metros de profundidad.

Figura 4.1: Cárama SJ4000 Wifi utilizana en los experimentos.

4.1.2. Software

Todos los algoritmos se han creado utilizando ROS bajo c++ y las librerías PCL y OpenCV.

- **ROS ("Robot Operating System")**

  ROS es un entorno de trabajo flexible para escribir software destinado a robots. Puede definirse como una colección de librerías que tienen el objetivo de simplificar la creación de complejos y robustos comportamientos para una gran variedad de plataformas.

ROS se creó con la intención de potenciar el desarrollo de software colaborativo para robots utilizando un diseño modular para que distintos grupos de desarrollo pudieran compartir y mezclar los módulos o paquetes que hayan desarrollado.

En mi caso, utilicé un paquete llamado "webcam_publisher"para obtener la imagen de la cámara y publicarla en un topic de ROS, este paquete fue cedido por el Laboratorio de Sistemas Inteligentes y personalmente añadí un pequeño algoritmo que generaba el archivo "times.txtçon el nombre de las imágenes, el tiempo con el que había sido capturado cada frame del vídeo y el tiempo de exposición de la cámara aunque por limitaciones de software no se pudo aplicar esta última parte. Por otra parte, también utilicé un paquete encontrado en la comunidad de ROS para suscribirme al topic donde se publicaban las imágenes y obtener todos los frames individuales para la ejecución de ROS.

- **OpenCV**

  OpenCV es una librería gratuita desarrollada originalmente por Intel con el objetivo de permitir la visión por computador en tiempo real.

  Los principales objetivos durante su desarrollo fueron crear una infraestructura muy optiimizada que permitiera tanto un uso sencillo como que fuera útil para desarrollos avanzados en la visió por computador de forma libre y disponible para el público.

  A día de hoy, OpenCV está disponible para los lenguajes de programación c, c++, Java, Python, Matlab y otros menos conocidas aportados por la comunidad como C Sharp o Haskell. Con los años se han desarrollado aplicaciones en áreas como reconocimiento facial y gesticular, rastreo de movimiento, robots móviles, incluso tiene librerías añadidas de aprendizaje para mejorar las apliaciones mencionadas anteriormente.

- **PCL ("Point Cloud Libraries")**

  PCL es un software independiente presentado en 2011 que permite utilizar algoritmos de generación y procesamiento de nubes de puntos tridimensionales que se utilizan en la visión por computador.

  Tiene otras aplicaciones como el filtrado de puntos, la comparación de diversas nubes de puntos o reconocimiento de superficies entre otras.

4.2. Experimentos

4.2.1. Implementación del algoritmo DSO:

En este apartado me gustaría comparar los resultados obtenido con el algoritmo DSO utilizando el "Monocular Visual Odometry Datasetz los obtenidos con mis propios conjuntos de imágenes para sacar conclusiones acerca de cómo mejorar el proyecto.

- **Entornos cerrados:**

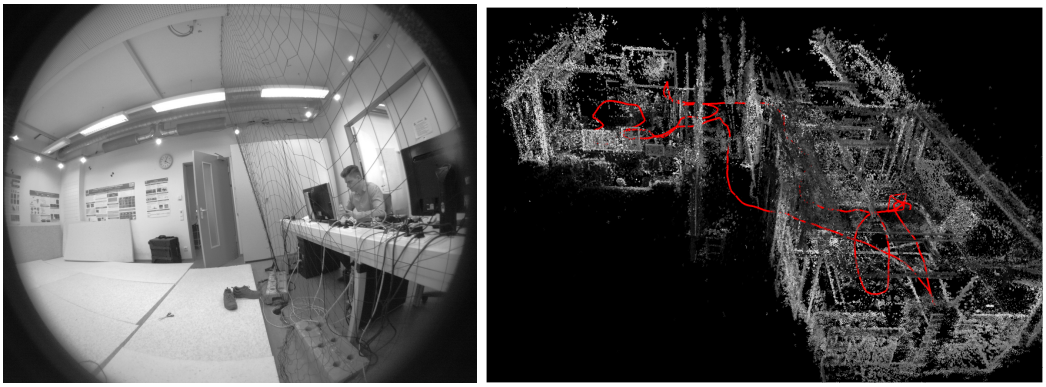

Figura 4.2: Imagen y mapa obtenidos de la primera secuencia. Fuente: Monocular Visual Odometry Dataset

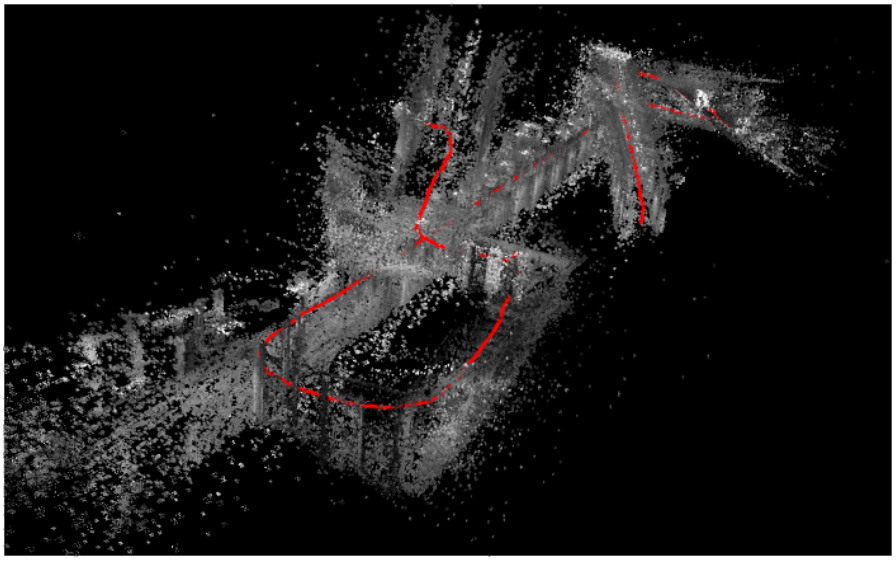

Figura 4.3: Nube de puntos de dos pasillos en la UC3M.

La Figura 4.3 es una nube de puntos obtenida a partir de un conjunto de imágenes grabadas en los pasillos de la UC3M, para ser más concretos en dos pasillos del mismo tamaño de la planta 3 y de la planta 1del edificio Agustín de Betancourt. Comienza y termina en la puerta del despacho 1.3.B16 y se puede comprobar como no es capaz de cerrar el bucle ni de hacer correctamente el ángulo en los giros como en la secuencia propuesta. También se puede comprobar el efecto del viñeteado, que da menos peso a los puntos más alejados del centro de la cámara y no genera tanto ruido detrás de las paredes y del techo.

- **Entornos abiertos:**

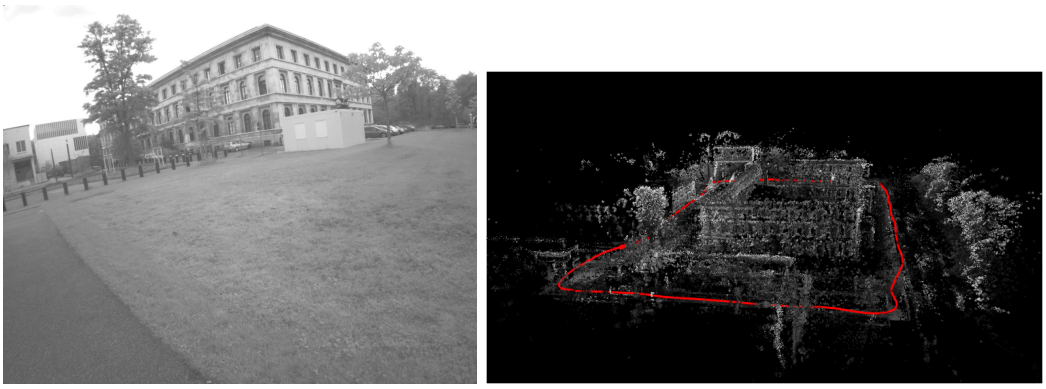

Figura 4.4: Imagen y mapa obtenidos de la quincuajésima secuencia. Fuente: Monocular Visual Odometry Dataset

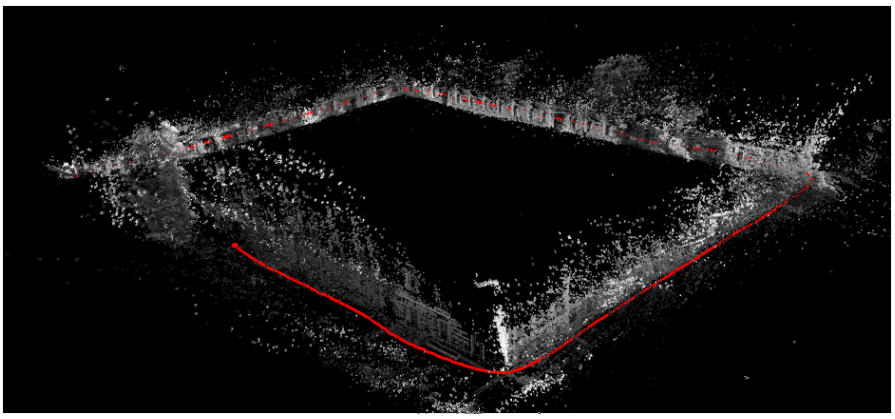

Figura 4.5: Nube de puntos generada alrededor del edificio Sabatini UC3M.

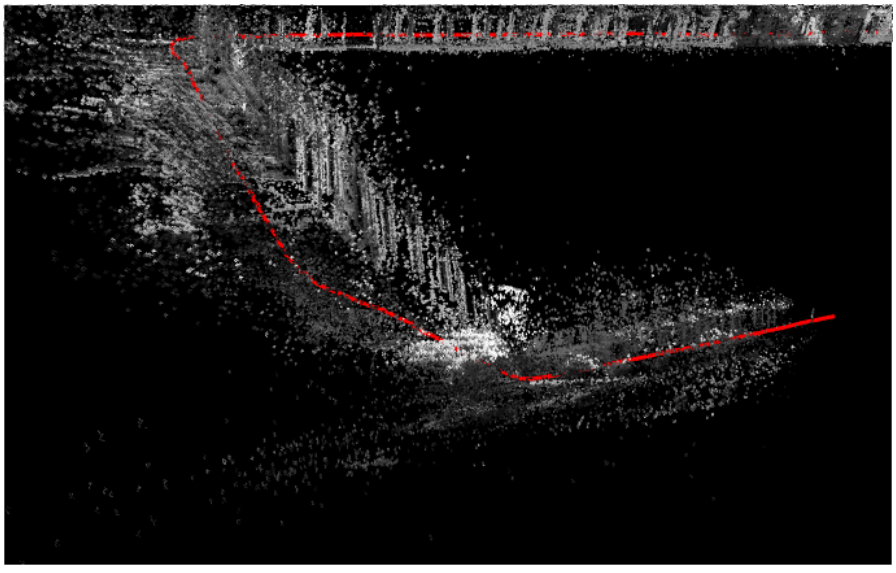

Figura 4.6: Error de escalado al girar en la primesa esquina de la Figura 4.5.

En entornos abierto, podemos comprobar como en este ejemplo sí ha realizado bien los giros de noventa grados en las esquinas (al contrario que la Figura 4.3) pero el error en la escala del mapa iba aumentando a cada esquina del edificio(Figura 4.6).

4.2.2. Obtención de la distancia:

Como he comentado antes, el propio algoritmo DSO genera un archivo de resultados donde podemos comprobar la distancia relativa adimensional calculada para cada keyframe. Este experimento se basa en generar las nubes de puntos de varios emplazamientos en linea recta con distancia conocida previamente para comprobar si se puede relacionar de forma directa con la distancia mediante un factor de conversión de 1 unidad adimensional a metros. Además, en las primeras localizaciones se han realizado alrededor de diez nubes de puntos (cinco en cada dirección) para realizar una media y comprobar la precisión y la repetibilidad del sistema.

- **Experimento 1: Pasillo de la universidad:**

Figura 4.7: Imagen de un pasillo del edificio Agustín de Betancourt en el campus de Leganés de la UC3M y una imagen a detalle de su correspondiente nube de puntos.

Este pasillo tiene 2.35 metros de ancho y realizé un recorrido de aproximadamente 44.7 metros en ambas direcciones y se han obtenido los siguientes resultados:

		Distancia obtenida				
Experimento	Distancia Recorrida	X	Y	Z	Módulo	Factor de Conversión
Pasillo UC3M	metros	Ud.	Ud.	Ud.	Ud.	metros
1_1	44,7	-2,083	-1,03	10,421	10,67693917	4,186593114
1_2	44,7	1,456	-2,967	14,781	15,14598911	2,951276387
2_1	44,7	-0,075	-0,48	16,827	16,83401182	2,655338518
2_2	44,7	-0,107	-0,48	16,827	16,8341848	2,655311233
3_1	44,7	-2,0476	-1,03	10,421	10,67008935	4,189280758
3_2	44,7	1,019	-2,352	9,2364	9,58547599	4,663305197
4_1	44,7	0,05664	-1,4314	17,54	17,59840089	2,540003508
4_2	44,7	0,19997	-0,711	9,5685	9,596963126	4,657723429
5_1	44,7	-2,1513	1,08399	9,3685	9,673257893	4,620987106
5_2	44,7	0,9065	-1,498	15,5489	15,64717347	2,856745987
Media					13,22624856	3,597656524
Desviación					3,444460376	0,934807374
% Desviación					26,04261034	25,98378606

Figura 4.8: Tabla de resultados del experimento 1 en el pasillo de la Universidad.

Donde el subíndice la dirección. La componente principal de la distancia está localizada en el eje Z y los ejes X e Y reflejan que las cámara no está firme o si el primer keyframe que se toma como referencia (0, 0, 0) está ligeramente inclinado.

Para la obtención del factor de conversión se ha dividido la distancia recorrida en metros entre el módulo adimensional (Ud.)la distancia obtenida generada por el algoritmo.

Los resultados obtenidos han sido un factor de conversión de 3,6 metros con una desviación típica del 26 %.

- **Experimento 2: Soportal 1**

  Los siguientes tres experimentos han sido realizados en la Urbanización donde resido, entre los edificios de la calle Concepción Arenal 3 y 5 en Fuenlabrada. Los dos primeros presentan características similares ya que se grabaron en el soportal del edificio en dos tramos paralelos separados por columnas, el tercero es el parque central de la urbanización, un entorno más abierto. Además, se ha utilizado la misma distancia para los tres de forma que sea más fácil de comparar.

Figura 4.9: Imagen del soportal y nube de puntos.

		Distancia obtenida				
Experimento	Distancia Recorrida	X	Y	Z	Módulo	Factor de Conversión
Soportal 1	metros	Ud.	Ud.	Ud.	Ud.	metros
1_1	36	-0,2165	0,2976	5,3014	5,314158444	6,774355785
1_2	36	-0,4114	0,6637	6,1897	6,238760593	5,770376898
2_1	36	-0,1751	0,5754	6,2223	6,251300861	5,758801376
2_2	36	-0,2229	0,6736	4,3073	4,365347026	8,246767046
3_1	36	-0,1563	0,5605	5,1016	5,134677448	7,011151209
3_2	36	-0,1738	0,6337	5,2482	5,289176247	6,806352884
4_1	36	0,0451	0,9575	5,2436	5,330495401	6,75359367
4_2	36	-0,293	0,9355	8,2087	8,267028785	4,354647956
5_1	36	-0,2959	0,6537	5,9857	6,028555796	5,971579465
5_2	36	-0,1612	0,4297	8,5784	8,590667849	4,190593867
Media					6,081016845	6,163822016
Desviación					1,364269248	1,23435061
% Desviación					22,43488684	20,02573414

Figura 4.10: Tabla de resultados del experimento 2 en el soportal de la urbanización.

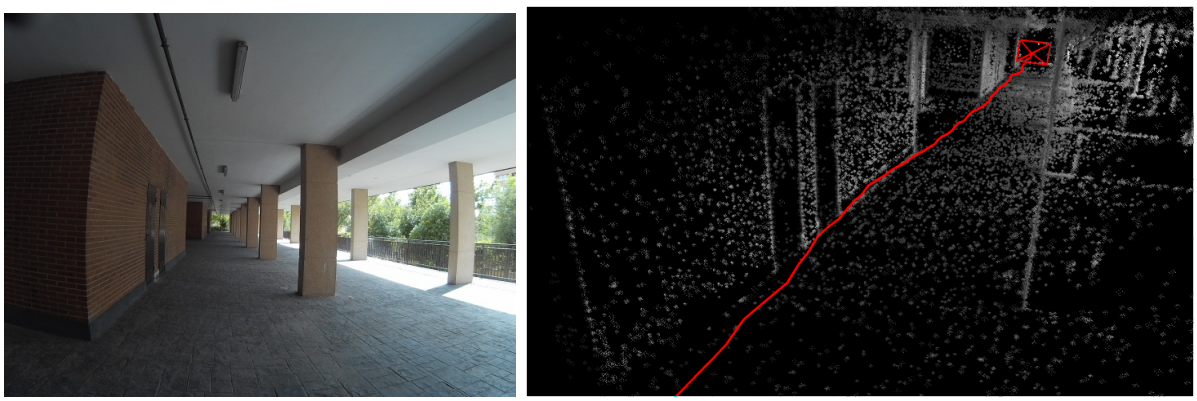

Figura 4.11: Imagen de otra parte del soportal y nube de puntos.

		Distancia obtenida				
Experimento	Distancia Recorrida	X	Y	Z	Módulo	Factor de Conversión
Soportal 2	metros	Ud.	Ud.	Ud.	Ud.	metros
1_1	36	-0,2427	0,3206	5,5054	5,520064928	6,52166242
1_2	36	-0,8213	0,0709	3,9562	4,041172966	8,908304669
2_1	36	-0,6708	-0,0094	9,7913	9,814255789	3,668133455
2_2	36	0,1516	0,8329	5,2105	5,278827069	6,819696787
3_1	36	0,3168	-0,9032	8,0194	8,076317777	4,457476909
3_2	36	0,3312	0,1308	6,7997	6,80901771	5,287106237
4_1	36	0,4264	0,1418	6,6046	6,619868984	5,438174092
4_2	36	0,1595	0,283	7,0986	7,106029215	5,06612046
5_1	36	0,1982	0,7166	6,7202	6,761204541	5,324495034
5_2	36	0,3329	0,6887	8,3313	8,366342677	4,302955472
Media					6,839310165	5,579412554
Desviación					1,661449049	1,508589318
% Desviación					24,29264076	27,0384974

Figura 4.12: Tabla de resultados del experimento 3 en el soportal de la urbanización.

Estos conjuntos de imágenes tienen aproximadamente 3,42m y 3,04. Se ha obtenido un factor de conversión de 6,16 para el primer conjunto y 5,57 para el segundo conjunto, con una desviación estándar de 20 % y 27 % respectivamente.

- **Parque central:**

Figura 4.13: Imagen del parque central y nube de puntos.

		Distancia obtenida				
Experimento	Distancia Recorrida	X	Y	Z	Módulo	Factor de Conversión
Parque Frontal	metros	Ud.	Ud.	Ud.	Ud.	metros
1_1	36	0,0256	-0,8591	3,8019	3,897839373	9,235885975
1_2	36	-0,0768	-0,167	3,4455	3,450399613	10,43357409
2_1	36				0	#¡DIV/0!
2_2	36	0,4619	-0,2498	4,0843	4,117919152	8,742279456
3_1	36	0,2867	0,0043	2,9099	2,923992714	12,31193218
3_2	36	0,1404	0	3,553	3,555772934	10,12438102
4_1	36	0,1669	0,0148	3,1394	3,14386816	11,45086186
4_2	36	0,2835	0,1507	3,063	3,079781119	11,68914238
5_1	36	0,1844	0,1087	2,879	2,886946492	12,46992284
5_2	36				0	#¡DIV/0!
Media					3,382064945	10,80724747
Desviación					0,454412405	1,391800059
% Desviación					13,43594557	12,87839537

Figura 4.14: Tabla de resultados del experimento 4 en el parque de la urbanización.

Este es el primer conjunto de grabaciones realizado en un entorno abierto. Las filas en blanco representan fallos en el programa cuando no es capaz de calcular la posición del siguiente frame, no se han tenido en cuenta para el cálculo de la media ni de la desviación. Los resultados obtenidos son 10,81 metros de factor de conversión y 13 % de desviación estándar.

- **Lateral exterior del edificio Sabatini:**

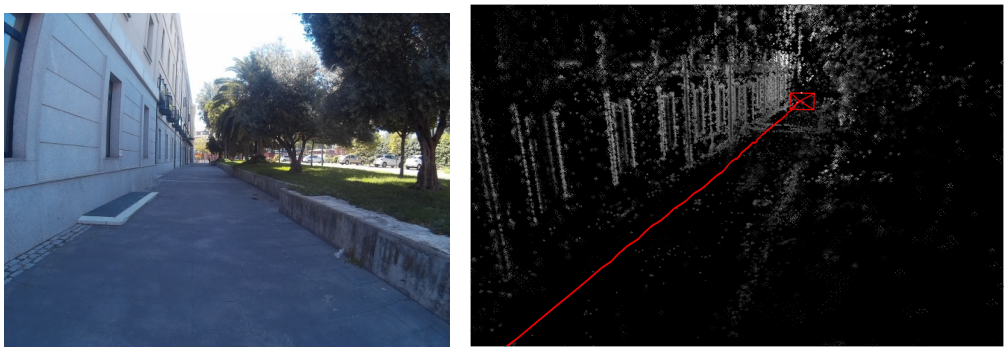

Figura 4.15: Imagen del lateral exterior (ala B) del edificio Sabatini en la UC3M y nube de puntos.

		Distancia obtenida				
Experimento	Distancia Recorrida	X	Y	Z	Módulo	Factor de Conversión
Lateral exterior	metros	Ud.	Ud.	Ud.	Ud.	metros
1_1	69,145	-0,7922	-0,8839	12,4495	12,50595499	5,528966003
1_2	69,145	-0,514	-0,4723	7,2282	7,261827492	9,521707872
2_1	69,145	-1,6473	-0,658	10,7509	10,89625679	6,3457572
2_2	69,145	0,1055	0,0523	8,1274	8,128252967	8,50674804
Media					9,698073062	7,475794779
Desviación					2,43032733	1,854341952
% Desviación					25,05989917	24,80461284

Figura 4.16: Tabla de resultados del experimento 5 en el lateral exterior del edificio Sabatini.

- **Lateral interior del edificio Sabatini:**

Figura 4.17: Imagen del lateral interior (ala C) del edificio Sabatini en la UC3M y nube de puntos.

		Distancia obtenida				
Experimento	Distancia Recorrida	X	Y	Z	Módulo	Factor de Conversión
Lateral interior	metros	Ud.	Ud.	Ud.	Ud.	metros
1_1	62,32	-0,208	0,4736	16,0772	16,08551898	3,874292156
1_2	62,32				0	#¡DIV/0!
2_1	62,32	-0,1863	0,6783	17,0028	17,01734428	3,662146041
2_2	62,32				0	#¡DIV/0!
Media					16,55143163	3,768219099
Desviación					0,658899991	0,150009956
% Desviación					3,980924466	3,980924466

Figura 4.18: Tabla de resultados del experimento 6 en el lateral interior ala C del edificio Sabatini.

En este experimento en concreto el programa tuvo problemas para generar la nube de puntos de la dirección contraria mostrada en la Figura 4.17 debido a que el programa presenta algunos problemas para inicializar en entornos cerrados.

- **Patio del edificio Sabatini:**

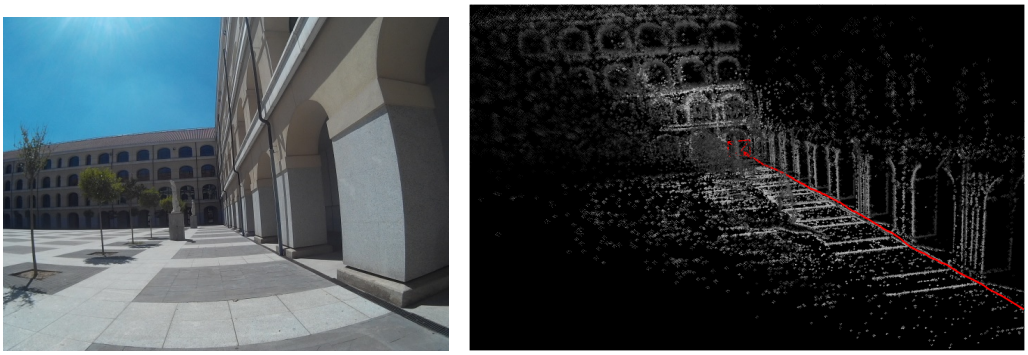

Figura 4.19: Imagen del patio del edificio Sabatini en la UC3M y nube de puntos.

		Distancia obtenida				
Experimento	Distancia Recorrida	X	Y	Z	Módulo	Factor de Conversión
Patio Sabatini	metros	Ud.	Ud.	Ud.	Ud.	metros
1_1	46,2	0,1192	0,9677	7,5959	7,658220991	6,032732675
1_2	46,2	0,5	0,4229	4,293	4,342659716	10,63864153
2_1	46,2	-0,2014	0,7154	5,3591	5,410389259	8,539126815
2_2	46,2	-0,2899	0,5785	4,7687	4,812401059	9,600197373
Media					5,555917756	8,702674598
Desviación					1,468067844	1,975588759
% Desviación					26,42349848	22,70093794

Figura 4.20: Tabla de resultados del experimento 7 en el patio del edificio Sabatini.

Se ha obtenido un factor de conversión de 7,48 para el lateral exterior, 3,76 para el lateral interior y 8,70 para el patio con unas desviaciones estándar de 24 %, 4 % y 22 % respectivamente.

Estos tres conjuntos de imágenes han sido realizados en la Universidad con la intención de comprobar el efecto de lo "estrecho" que sea el entorno en la distancia adimensional calculada por el algoritmo. Teniendo un entorno muy cerrado (Figura **??**) similar al pasillo de la Universidad del primer experimento (Figura **??**), un entorno algo más abierto (Figura **??**) similar a dos de los experimentos dentro de la urbanización (Fugura **??**) y un entorno abierto (Figura **??**) similar al experimento en el parque (Figura **??**).

Experimento nº	Localización	"Anchura" [m]	Factor de Conversión [m]
1	Pasillo UC3M	2,35	3,6
2	Soportal 1	3,42	6,16
3	Soportal 2	3,04	5,56
4	Parque Frontal	"abierto"	10,81
5	Lateral exterior	4,75	7,48
6	Lateral interior	2,75	3,77
7	Patio Sabatini	"abierto"	8,7

Figura 4.21: Tabla de comparación del factor de conversión y la anchura.

Podemos comprobar como aumenta el factor de conversión cuanto más abierto es un escenario.

4.2.3. Nube de puntos recoloreada:

La última mejora propuesta al sistema era la obtención de un entorno tridimensional en el que cada uno de sus puntos tuviera el color que le corresponde en la imagen original, de forma que se pareciera un poco más a la imagen real.

Figura 4.22: Imagen de pasillo del Edificio Agustín de Betancourt y su nube de puntos a color.

Figura 4.23: Imagen del soportal y su nube de puntos a color.

Figura 4.24: Imagen del parque central y su nube de puntos a color.

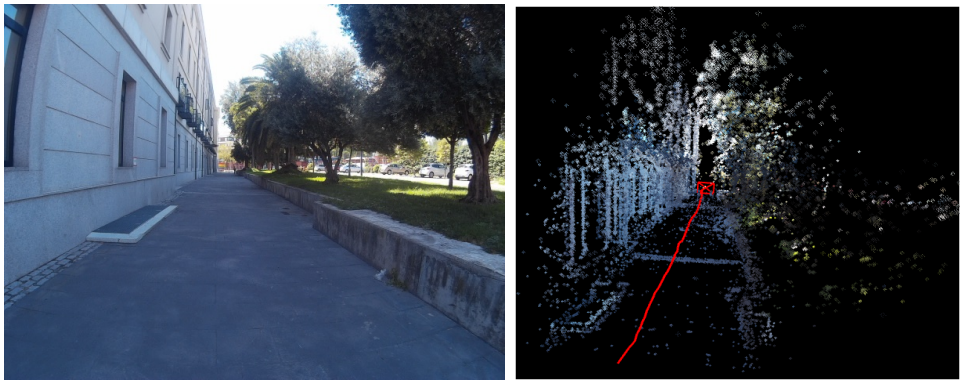

Figura 4.25: Imagen de pasillo del exterior del edificio Sabatini UC3M (ala B) y su nube de puntos a color.

Capítulo 5

Conclusiones

La principal conclusión que se puede obtener de este proyecto es que no he sido capaz de generar una nube de puntos utilizando la implementación del algoritmo DSO tan precisa como las generadas con el dataset propuesto por sus autores. La principal diferencia entre los conjuntos era la calibración de la cámara, más concretamente la imposibilidad de acceder al tiempo de exposición, la falta del ejecutable para realizar el viñeteado y la diferencia de framerate, alrededor del doble para el conjunto propuesto.

Sobre el factor de conversión para obtener la distancia, se ha demostrados con los experimentos realizados anteriormente no es posible determinar la distancia únicamente utilizando la información recibida por la cámara, ya que es demasiado variable en distintos entornos e incluso dentro del mismo entorno, tiene una gran desviación estándar y carece de repetibilidad. Aún así podríamos clasificar los coeficientes obtenido de forma que el algoritmo calcula distancias menores para entornos más abiertos.

La nube de puntos con sus colores originales a pesar de ser más vistoso, apenas presenta una mejora en la comprensión. Además, el tiempo de procesado es demasiado alto ya que tiene que buscar y abrir cada imagen para obtener el color RGB del píxel concreto y un conjunto de imágenes sencillo como el de la Figura **??** contiene alrededor de 2.25 millones de puntos. Aunque me hubiera gustado generar un una nube de puntos de uno de los datasets propuestos por los autores y haberla comparado con la generada automáticamente en escala de grises.

Capítulo 6

Trabajos futuros

Como trabajo futuro se propone utilizar otra cámara que se pueda calibrar correctamente para las exigencias del algoritmo DSO y comprobar que se puedan reproducir nubes de puntos similares a las generadas con el conjunto de datos propuesto por los autores.

También se propone la implementación en un modulo de ROS que funcione con imágenes obtenidas de un video en tiempo real. Que pueda ser utilizado en un dron con otras funciones de forma simultánea y que permita una fusión sensorial para obtener la distancia recorrida por el dron y mejorar el rendimiento y la precisión del algoritmo al eliminar el cálculo de la posición a través de las imágenes.

Apéndice

Apéndice A. Marco Legislativo

A continuación se va a enumerar la legislación que afecta al vuelo de drones y a la grabación de imágenes en exteriores que puede afectar a nuestro proyecto:

1. Real Decreto RPAS 27-10-2016 por el que se regula la utilización de aeronaves por control remoto.

Apéndice B. Aspecto Socio-Económico

CÓDIGO	UNIDAD	DESCRIPCIÓN	MEDICIÓN	PRECIO UNITARIO	PRECIO TOTAL
		CAPÍTULO 1: HARDWARE			
1.01	Ud.	**Ordenador portátil Acer Aspire F 15 :** Ordenador portátil marca Acer con microprocesador intel i5-6200U 2,3Ghz, tarjeta gráfica NVIDIA GeForce 920M con 2Gb de memoria VRAM dedicada, 8Gb de memoria RAM DDR3 y 1 Tb de disco duro. Utilizado para el procesamiento de la nube de puntos y la programación de los algoritmos. Utiliza un doble arranque del sistema operativo libre Linux con distribución Ubuntu 16.04 y Windows 10 con clave incluida en el precio. El SO Windows 10 es necesario para el uso de Microsoft Office 2016 y el SO Ubuntu 16,04 es necesario para el uso de ROS y del algoritmo DSO. Totalmente configurado probado y en funcionamiento.	1	600,00	600,00
1.02	Ud.	**Cámara SJ4000 Wifi:** Suministro de la cámara deportiva SJ4000 Wifi. Presenta un objetivo de 12 Megapíxeles y sensor CMOS, lente de gran angular de 170º. Empleada para la captación de imágenes por computador para la posterior recreación de entornos en tres dimensiones. Totalmente probado y en funcionamiento.	1	99,00	99,00
		Precio total capítulo 1:			699,00
		CAPÍTULO 2: HORAS DE TRABAJO			
2.01	Hora	**Alumno:** Número de horas empleadas por el alumno para la realización de este proyecto, la asquisición de los fundamentos teóricos, el diseño de los algoritmos, los experimentos realizados y la redacción del trabajo.	300	20,00	6.000,00
2.02	Hora	**Tutor:** Número de horas del ingeniero tutor del Trabajo de Fin de Grado dedicadas al asesoramiento del alumno vía tutorías.	30	32,00	960,00
		Precio total capítulo 2:			6.960,00
		PRECIO TOTAL:			7.659,00

Figura 1: Presupuesto del proyecto.

Bibliografía

[1] "Ros.org. (2017). camera_calibration," http://wiki.ros.org/camera_calibration, [Accessed 25 Sep. 2017].

[2] J. Engel, V. Usenko, and D. Cremers, "A photometrically calibrated benchmark for monocular visual odometry," in *arXiv:1607.02555*, July 2016.

[3] J. Engel, V. Koltun, and D. Cremers, "Direct sparse odometry," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PP, no. 99, pp. 1–1, 2017.

[4] C. Wheatstone, "Contributions to the physiology of vision.–part the first. on some remarkable, and hitherto unobserved, phenomena of binocular vision," *Philosophical Transactions of the Royal Society of London*, vol. 128, pp. 371–394, 1838. [Online]. Available: http://www.jstor.org/stable/108203

[5] J. Civera, A. J. Davison, and J. M. M. Montiel, "Inverse depth parametrization for monocular slam," *IEEE Transactions on Robotics*, vol. 24, no. 5, pp. 932–945, Oct 2008.

[6] H. Strasdat, J. Montiel, and A. J. Davison, "Scale drift-aware large scale monocular slam," *Robotics: Science and Systems VI*, vol. 2, 2010.

[7] J. Engel, T. Schöps, and D. Cremers, *LSD-SLAM: Large-Scale Direct Monocular SLAM*. Cham: Springer International Publishing, 2014, pp. 834–849. [Online]. Available: https://doi.org/10.1007/978-3-319-10605-2_54

[8] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardós, "Orb-slam: A versatile and accurate monocular slam system," *IEEE Transactions on Robotics*, vol. 31, no. 5, pp. 1147–1163, Oct 2015.

[9] J. Sánchez Pérez, "Visión tridimensional."

[10] R. Hartley and A. Zisserman, "Multiple view geometry in computer vision," *Robotica*, vol. 23, no. 2, pp. 271–271, 2005.

[11] C. Zhang and Z. Zhang, *Calibration Between Depth and Color Sensors for Commodity Depth Cameras*. Cham: Springer International Publishing, 2014, pp. 47–64. [Online]. Available: https://doi.org/10.1007/978-3-319-08651-4_3

[12] "Docs.opencv.org. (2017). feature detection and description — opencv 3.0.0-dev documentation." http://docs.opencv.org/3.0-beta/doc/py_tutorials/

py_feature2d/py_table_of_contents_feature2d/py_table_of_contents_feature2d.html, [Accessed 24 Sep. 2017].